Below is a Comparison of C and C++ with program examples to illustrate the key differences:-

---

1. Basic Syntax and Structure:

C Program Example:

c code:-

```
#include <stdio.h>

int main() {
    printf("Hello, World!\n");
    return 0;
}
```

- **Explanation**: In C, you include the standard header `<stdio.h>` to use input/output functions like `printf()`. The program consists of a single `main()` function, which returns an integer.

C++ Program Example:

cpp code :-

```
#include <iostream>

int main() {
    std::cout << "Hello, World!" << std::endl;
    return 0;
}
```

- **Explanation**: In C++, instead of `<stdio.h>`, we include `<iostream>`, and the output is printed using `std::cout`. C++ uses the `iostream` library for input/output operations, and we use `std::endl` for a newline.

---

2. Object-Oriented Programming (OOP) Support:

C Program Example (No OOP):

In C, we don't have classes or objects.

C code:-

```
#include <stdio.h>

struct Rectangle {
    int width;
    int height;
};
```

```
int main() {
    struct Rectangle rect;
    rect.width = 10;
    rect.height = 5;
    printf("Area: %d\n", rect.width * rect.height);
    return 0;
}
```

- **Explanation**: We use a `struct` to define a `Rectangle`, but it has no methods (functions). In C, data and functions are separate.

C++ Program Example (With OOP):-

In C++, we can use **classes** to group data and functions together.

cpp code:-

```
#include <iostream>
using namespace std;

class Rectangle {
public:
    int width;
    int height;

    int area() {
        return width * height;
    }
};

int main() {
    Rectangle rect;
    rect.width = 10;
    rect.height = 5;
    cout << "Area: " << rect.area() << endl;
    return 0;
}
```

- **Explanation**: In C++, the `Rectangle` is defined as a class, and we can add a method `area()` that operates on the class's data members (`width` and `height`).

---

3. Function Overloading:

C Program Example (No Overloading):

In C, each function must have a unique name if it is to perform different tasks.

C code:-

```
#include <stdio.h>

void printSum(int a, int b) {
    printf("Sum: %d\n", a + b);
}

int main() {
    printSum(10, 20);
    return 0;
}
```

- **Explanation**: We define a function `printSum()` that accepts two integers and prints their sum. In C, if you wanted to handle different types (e.g., float), you would need separate functions.

C++ Program Example (With Overloading):

In C++, function overloading allows multiple functions with the same name but different parameters.

cpp code:-

```
#include <iostream>
using namespace std;

void printSum(int a, int b) {
    cout << "Sum of integers: " << a + b << endl;
}

void printSum(double a, double b) {
    cout << "Sum of doubles: " << a + b << endl;
}

int main() {
    printSum(10, 20);
    printSum(10.5, 20.5);
    return 0;
}
```

- **Explanation**: C++ allows function overloading. The function `printSum()` is defined twice with different parameter types, and the compiler will choose the correct one based on the arguments provided.

---

4. Memory Management:

C Program Example (Manual Memory Management with `malloc`):

In C, you manage memory manually using `malloc()` and `free()`.

```
c code
#include <stdio.h>
#include <stdlib.h>

int main() {
    int *arr = (int *)malloc(5 * sizeof(int)); // Allocate memory for an
array of 5 integers

    if (arr == NULL) {
        printf("Memory allocation failed.\n");
        return 1;
    }

    for (int i = 0; i < 5; i++) {
        arr[i] = i + 1;  // Assign values to the array
    }

    for (int i = 0; i < 5; i++) {
        printf("%d ", arr[i]);  // Print the array elements
    }

    free(arr);  // Free the allocated memory
    return 0;
}
```

- **Explanation**: In C, we manually allocate and deallocate memory using `malloc()` and `free()`.

C++ Program Example (With `new` and `delete`):

In C++, memory management is done with `new` and `delete`.

```
cpp code
#include <iostream>
using namespace std;

int main() {
```

```
    int* arr = new int[5]; // Allocate memory for an array of 5 integers

    for (int i = 0; i < 5; i++) {
        arr[i] = i + 1;
    }

    for (int i = 0; i < 5; i++) {
        cout << arr[i] << " ";
    }

    delete[] arr; // Free the allocated memory
    return 0;
}
```

- **Explanation**: In C++, we use `new` to allocate memory and `delete[]` to deallocate it. C++ also supports dynamic memory allocation with classes and objects.

---

5. Exception Handling:

C Program Example (No Exception Handling):

In C, errors are handled using return values and error codes.

```
C code
#include <stdio.h>

int divide(int a, int b) {
    if (b == 0) {
        return -1;  // Error code for division by zero
    }
    return a / b;
}

int main() {
    int result = divide(10, 0);
    if (result == -1) {
        printf("Error: Division by zero\n");
    } else {
        printf("Result: %d\n", result);
    }
    return 0;
}
```

- **Explanation**: In C, we manually check for errors by returning error codes (e.g., `-1` for division by zero).

C++ Program Example (With Exception Handling):

In C++, we can use `try`, `catch`, and `throw` to handle exceptions.

cpp code:-
```
#include <iostream>
#include <stdexcept>
using namespace std;

int divide(int a, int b) {
    if (b == 0) {
        throw runtime_error("Error: Division by zero");
    }
    return a / b;
}

int main() {
    try {
        int result = divide(10, 0);
        cout << "Result: " << result << endl;
    } catch (const runtime_error& e) {
        cout << e.what() << endl;
    }
    return 0;
}
```

- **Explanation**: In C++, we use exceptions to handle errors. The `throw` keyword is used to signal an error, and `catch` is used to handle it in a structured way.

---

6. Input/Output:-

C Program Example (Using `scanf` and `printf`):

C code:-
```
#include <stdio.h>

int main() {
    int a;
    printf("Enter a number: ");
    scanf("%d", &a);  // User input
    printf("You entered: %d\n", a);
```

```
    return 0;
}
```

- **Explanation**: In C, we use `scanf()` for input and `printf()` for output.

C++ Program Example (Using `cin` and `cout`):

cpp code:-

```
#include <iostream>
using namespace std;

int main() {
    int a;
    cout << "Enter a number: ";
    cin >> a;  // User input
    cout << "You entered: " << a << endl;
    return 0;
}
```

- **Explanation**: In C++, we use `cin` for input and `cout` for output, which are part of the `iostream` library.

Summary of Key Differences in Code:

Feature	C Example	C++ Example
Output	`printf()`	`std::cout`
Input	`scanf()`	`std::cin`
Memory Management	`malloc()` and `free()`	`new` and `delete`
OOP Support	No classes or objects	Classes, Objects, and Methods
Function Overloading	Not Supported	Supported
Exception Handling	Error codes	`try`, `catch`, and `throw` for exceptions
Data Structure	`struct` without methods	`class` with both data and methods

---

Conclusion:

- **C** is simpler, more low-level, and focused on procedural programming.
- **C++** is a more advanced language that supports object-oriented programming, function overloading, exception handling, and more modern features like the Standard Template Library (STL), making it suitable for large-scale applications.